7-2006

Simulated Annealing with min-cut and greedy perturbations

Brian J. Cody

Simulated Annealing with Min-Cut and Greedy Perturbations

by

Brian J Cody

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Assistant Professor of Computer Engineering Dr. Marcin Łukowiak
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
July 2006

Approved By:

Marcin Łukowiak

Dr. Marcin Łukowiak
Assistant Professor of Computer Engineering
Primary Advisor

Stanisław Radziszowski

Dr. Stanisław Radziszowski
Professor of Computer Science

Pratapa V. Reddy

Dr. Pratapa V. Reddy
Professor of Computer Engineering

Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

Title: Simulated Annealing with Min-Cut and Greedy Perturbations

I, Brian J Cody, hereby grant permission to the Wallace Memorial Library reporduce my thesis in whole or part.

Brian Cody

Brian J Cody

7/29/06

Date

Dedication

This thesis is dedicated to my parents, Howard and Pat, and to April, for their support that made this possible.

Acknowledgments

I'd like to acknowledge Dr. Marcin Łukowiak for his supervision and dedication during this work. I would also like to thank my committee members Dr. Stanisław Radziszowski and Dr. Pratapa V. Reddy for their support and guidance.

Abstract

Custom integrated circuit design requires an ever increasing number of elements to be placed on a physical die. The process of searching for an optimal solution is NP-hard so heuristics are required to achieve satisfactory results under time constraints.

Simulated Annealing is an algorithm which uses randomly generated perturbations to adjust a single solution. The effect of a generated perturbation is examined by a cost function which evaluates the solution. If the perturbation decreases the cost, it is accepted. If it increases the cost, it is accepted probabilistically. Such an approach allows the algorithm to avoid local minima and find satisfactory solutions. One problem faced by Simulated Annealing is that it can take a very large number of iterations to reach a desired result. Greedy perturbations use knowledge of the system to generate solutions which may be satisfactory after fewer iterations than non-greedy, however previous work has indicated that the exclusive use of greedy perturbations seems to result in a solution constrained to local minima.

Min-cut is a procedure in which a graph is split into two pieces with the least interconnection possible between them. Using this with a placement problem helps to recognize components which belong to the same functional unit and thus enhance results of Simulated Annealing. The feasibility of this approach has been assessed.

Hardware, through parallelization, can be used to increase the performance of algorithms by decreasing runtime. The possibility of increased performance motivated the exploration of the ability to model greedy perturbations in hardware. The use of greedy perturbations while avoiding local minima was also explored.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The integrated circuit (IC) production industry has been driven by ever increasing consumer demand for faster and more complex products which has come from a number of sources including consumer, military, and research applications. This demand has prompted the technological advances (and refinement) in the fabrication process which allows production of higher transistor density chips. With a higher density in chips, there are more transistors that need to be placed and routed on the semiconductor die. The task of finding the optimal placement and routing is NP-hard and computer-assisted design (CAD) is a requirement [13]. Also as companies compete to deliver the products the popular belief is that the first to market is usually adopted as a standard and produces far more revenue than later arrivals. With this in mind, companies must have a scalable development model that allows them to quickly release these increasingly complex products. There are three general design styles for integrated circuitry. One is full-custom, which involves individually placing transistors to maximize the efficiency of the space of the layout. Full-custom allows a design to have a lot of power optimization and fine component control to meet other constraints, but it is at the expense of design time. Programmable Logic Devices (PLD's) are at the other end of the design spectrum where all the hardware already exists and a logical design is fitted into it. While this approach results in much faster design time, it is much less efficient in terms of size and space utilization. In a semi-custom (or standard cell) approach, basic logic gates (or cells) are pre-constructed and used in designing complete circuits. This is a compromise between the two previous design styles by allowing significant customization

however reducing the placement and routing problem from the transistor level to the level of a gate. The standard cell approach is used in this work.

When using any of the design styles, one important step of the design process is placement. In placement, the physical components which make up the IC's functionality are given physical locations where they will be placed on a semiconductor die. Placement's importance comes from the fact that the interconnect distances between components joined by nets determine the wire length of the design which influences characteristics such as chip timing, power use, and power distribution.

A difficulty with design placement comes from the size of the designs being placed. An exhaustive search of a design of just ten components and ten spaces where any component can enter any space requires an evaluation of 3,628,800 *(n!)* permutations. Modern design placements deal with tens to hundreds of thousands of components which are not fit into slots, but into fine granularity positions with extra space (or *white space*). The number of possible positions in these designs makes an exhaustive search impossible and the task of finding the optimal placement pattern an unrealistic pursuit [11]. Instead, simply an acceptable solution is sought where an acceptable solution is one that is not necessarily optimal. There is a large number of acceptable solutions to most design placements given the conditions (or constraints) that define the design. It is only necessary to find an acceptable solution for the placement to be considered solved.

It is necessary to use a more advanced algorithm, such as one employing heuristics, since it isn't realistic to perform an exhaustive search to find a solution. Heuristics intelligently direct the attention of the solving mechanism such that an acceptable solution is more quickly found. Simulated Annealing (SA) is one such heuristic [11]. SA models the process of annealing in metallurgy to solve combinational problems. In traditional annealing, a metal is treated to a very high temperature followed by a cooling bath. While this happens, the molecules in the metal are at first very excited because of the heat and they start moving around. Once the cooling begins the molecules slow down and form in a more uniform pattern. This reduces the defects in the metal making it less brittle. The concept

of Simulated Annealing is that this process can be translated into the realm of integrated circuitry, where the cells in a design model the molecules in a metal undead an annealing process [11].

Standard SA uses purely random modifications (or perturbations) to a solution in an attempt to improve it. These random perturbations are accepted probabilistically: if they appear to lead to a better solution they are accepted, if they appear to lead to a worse solution they are tested against a probability that depends on the simulated temperature of the system and how much worse they make the solution. The value of a solution is judged by the *cost function* which typically uses qualities such as net wire length and component overlap.

Greedy perturbations are those which are not entirely random, but rather directed in such a way as to converge to a solution quicker. For example, normally when a component is selected for a perturbation it is given a new random location without any regard for any known information. With a greedy perturbation such as the force directed technique [9], the component is given a random location with regard to its connections to other components in an attempt to create a better perturbation. While these do tend to create runs which converge faster, they also tend to converge to a worse result because the placement gets stuck in a local minimum. With this in mind it has previous been shown that a mix of random and greedy perturbations produces favorable results quicker [9].

This work has two separate parts with the goal of improving the results of design placement. The parts have been broken into software enhancement of SA and hardware modeling. In the software enhancement part, a min-cut algorithm has been developed to attempt to find the groupings for the given input circuit by connectedness alone. A feasibility analysis has been conducted with cost/benefit in mind. Greedy perturbations have been introduced to an SA application and their ability to avoid local minima has been evaluated. The hardware modeled portion is a part of a VHDL implementation of the SA algorithm. A prediction for the speedup realized by a hardware implementation is given and justified with software simulations.

1.1 Motivation

Consumer and industry drive has motivated further work on the placement problem. These drives come from industry competition for better products and consumer demand for more advanced products. Adding more features means adding more transistors which prompts the need for more efficient EDA (Electronic Design Automation) algorithms including placement algorithms. The need for faster designs brings with it a need for smaller transistor technology, which while it doesn't directly imply more transistors, it allows higher densities. It is believed that decreasing design time will increase profits in the long run by establishing market.

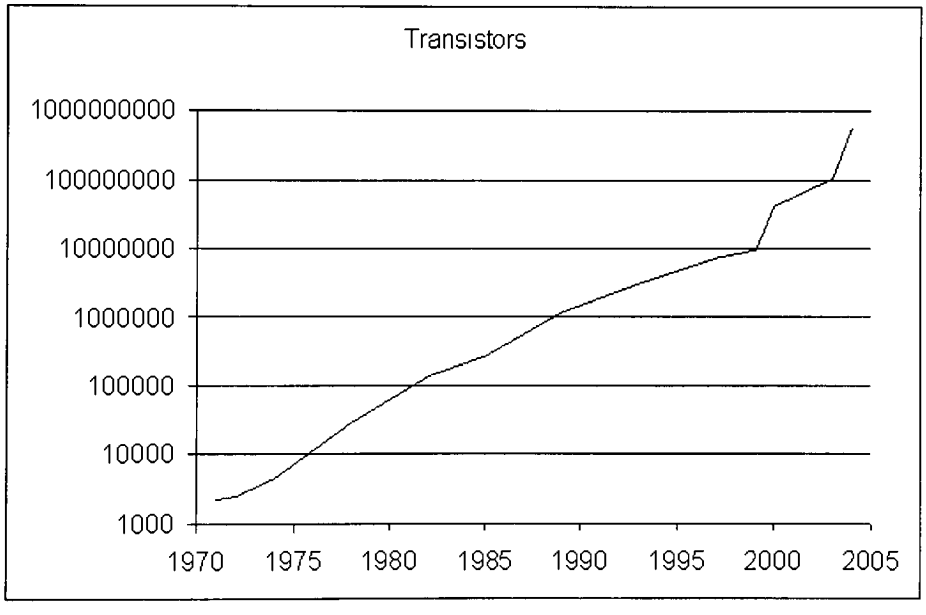

Figure 1.1: The number of transistors in Intel PC processors by year

On the consumer side there is a consistent demand for increasingly powerful products as software applications are becoming more complex. These more powerful products are generally realized through an increase in transistor count. In Figure 1.1 the transistor counts in Intel PC processors from 1971 to 2004 are shown [8]. It is clear that in at least this case the designs are becoming exponentially more complex.

1.2 Prior work

The placement problem has been approached in several ways. One technique involves alternating between two algorithms during the placement process. The placer, GORDIAN [12], acts on standard cell designs by moves between global optimization and partitioning steps. Optimization starts with an initial region the size of the design with all the components residing within it. Quadratic programming is used to model nets as rubber bands when determining placement. Component groups called slices are iteratively made, dividing the region into sub regions as optimization runs at each level. These slices attempt to create partitions of equal numbers of components.

Dragon2000 [20] is a standard cell placement tool which utilizes a top-down hierarchical algorithm. There are two phases to the program: global placement and detailed placement. In global placement, cells are placed in one of a few global bins depending on their estimated location. The estimation comes from the desire to lower wire length while keeping the cells spread out. These few global bins are then broken up into smaller bins and each cell gets assigned a new smaller bin inside its original larger bin. This repeats until a bin contains no more than seven cells, after which it is no longer broken up. Final placement is done on the last set of sub-bins to eliminate overlap since wirelength has been largely addressed by the bin assignment.

Prior to this thesis, the program FastPlace [19] was made with the claim to be much faster than all previous placers that were found to benchmark against. The algorithm involves a three-stage optimization process which begins with general global optimization. The first stage is a coarse level optimization which continues until the design has an equal distribution of components across its area. Optimization is done by modeling the potential energy of nets as if there were physical springs bounding cells together. The second stage performs more detailed global optimization with iterative local refinement as well. Iterative local refinement is done by assigning bins to cells, and calculating the cost of moving a cell from one bin to one of its four neighbors. The last stage uses a greedy heuristic to further reduce wire length and legalizes the design placement (eliminates overlap).

Chapter 2

Background

2.1 Placement

Placement itself is a general problem that exists in different industries. The defining characteristic is that a number of components are being fit into a finite amount of space with rules regarding the desired outcome. In standard cell EDA the components are logic gates which must be placed non-overlapping on a semiconductor die. These components are connected with nets (which later must be routed). It is highly desirable to reduce the lengths of these nets as excessive wire length increases power consumption, wiring congestion, timing delay, and complicates manufacturability [9].

Placement is made into a minimization problem when solutions are represented by a cost function. A cost function places a value on a given solution so that it can be compared directly to other solutions. It is derived with a number of parameters from the solution depending on implementation. The goal of the cost function is to direct an algorithm to create better solutions. If minimizing the cost function does not produce an acceptable solution, then the cost function is not satisfactory. In practice minimizing the cost function is NP-hard as the number of parameters which make it up can be hundreds to hundreds of thousands.

2.2 Annealing and Simulated Annealing

Simulated Annealing is a metaheuristic based on metallurgical annealing. To anneal means to heat and then cool a material (usually metal) to soften it and make it less brittle. This works by heating the material being annealed to allow its molecules to more move freely. When the temperature lowers, the object begins to crystallize into a regular structure of lower energy. In a lower energy state, there is less free movement of atoms. Theoretically if the initial heat is high enough and if the object is cooled slowly enough, it will reach a state of lowest energy possible [6]. In practice a near-lowest energy state will be reached instead, with a few points of high energy (or defects) [3]. The states of lowest energy and near-lowest energy are analogous to the global solution and local minima of a combinational problem, respectively. By modeling this energy as a cost function, Simulated Annealing applies this idea to minimization problems.

2.2.1 Cost function

At the core of the Simulated Annealing process is a cost function which evaluates the current solution state. This cost function generates a single value to describe the quality of the state so that it can be compared to other states. The terms that enter the cost function vary between implementations. In work on mixing random and greedy perturbations [9] and stochastic optimization [17], only wire length was considered. Wire congestion was measured for record in these examples, but not used. The general cost function is given in Equation 2.1. The α values are constant scaling factors for the parameters β. i iterates across the range of α scaling factors and j iterates across the range of β parameters.

$$F = \Sigma_{i,j} \left(\alpha_i \cdot \beta_j\right) \tag{2.1}$$

$$F = \left(\beta_{wl} + \alpha_{ol} \cdot \beta_{ol}\right) \tag{2.2}$$

In this thesis two parameters (β) are used in the cost function. The first is wire length because in general with a decrease of the wire length of a design, there is a decrease in

the routing congestion, overall timing, and power use [9]. It is the general metric by which placement tools are compared, so wire length is measured similarly between tools. Another important aspect of the solution created is that it must be legal. A legal solution cannot have any overlap between components because the fabrication process does not allow it; therefore overlap is also a parameter in the cost function. Typically the amount of overlap is multiplied by a large overlap factor to make sure that it gets fixed. The cost function as used in this work appears in Equation 2.2, where β_{wl} is the wire length of the system and α_{ol} and β_{ol} are the overlap scaling factor and overlap quantity respectively. In a design that contains a thousand cells and a thousand nets, the total number of parameters in the cost function is over five-hundred thousand. That's one parameter per net, and one per possible overlap. Since this is prohibitive to calculate at every iteration of the application, a *delta cost function* is used, which looks at only the values which have changed, and produces a delta value.

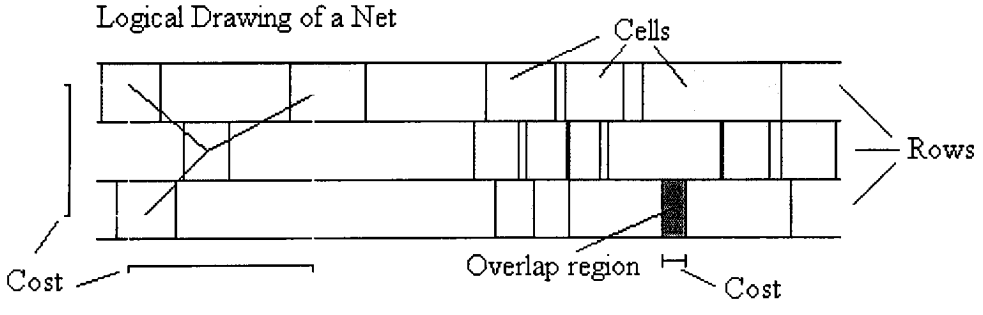

Figure 2.1: A demonstration of a net's cost and an overlap's cost

Wire length is commonly computed as the width and the height of the minimum bounding box of a given net (also known as the half-perimeter) as shown in Figure 2.1. This can be weighted vertically or horizontally to fit the design. For example if a design is twice as tall as it is wide (a very common occurrence in a datapath design) vertical routing space costs a lot more than horizontal routing space. Also shown in Figure 2.1 is an overlap condition. The cost for an overlap is the square of the overlap width (in a row-based system), plus an offset, multiplied by a constant. The purpose of the high offset is to force the design

to eliminate overlap once the temperature has lowered and a solution is about to be reached. If this is not done, then the overlap cost would be shadowed by the wire length cost, and consequently the overlap would not be fixed [11].

2.2.2 Probabilistic criterion

The Boltzmann equation (Equation 2.3) describes the probability that a state (or solution) of the system is at equilibrium (or a minimum) at a given temperature and energy level [5]. T represents the current temperature, E the current energy in the system and A and k are constants. It is very difficult to determine if a given state is in fact the equilibrium state without knowing the equilibrium energy of the system. This wouldn't be known without the global minimum solution (and if that was known there'd be no placement to be done!). However using this probability it can be determined if one state has a better chance of being in equilibrium than another.

$$f(E) = \frac{1}{A \exp E/kT} \tag{2.3}$$

The relative probability between two states can be found with a ratio of probabilities which can be evaluated between the current state and a given generated state with their temperatures and energy levels. This ratio (2.4) provides a means to determine if a change is to be accepted probabilistically. The A constant is canceled out in the ratio and for simplicity the k constant is absorbed by T. Since this is only a model of annealing, the temperature is unit-less. From the equation, if a change will result in a lower energy level, then the probability that it will be accepted is one because it has a better chance to be at equilibrium. Under this condition it's automatically accepted. If the change will result in a higher energy level, then the ratio gives a probability about whether or not to accept the change.

$$P(\Delta E) = min \left\{ \exp \frac{-\Delta E}{T}, 1 \right\} \tag{2.4}$$

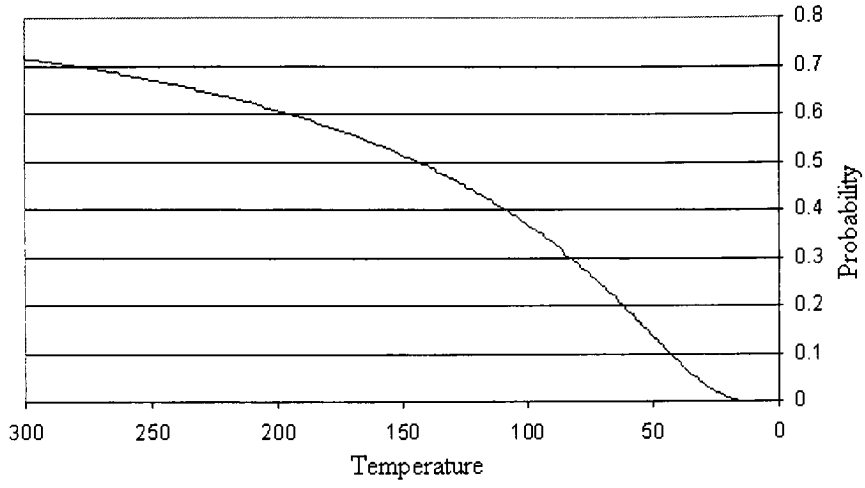

Figure 2.2: Probability that a move is accepted at temperatures

For example in Figure 2.2, the probability for acceptance is shown for a move which increases the solution cost by a constant. This probability changes across temperatures by Equation 2.4. At the higher temperatures the probability is very close to one (while the graph shows up to 300 degrees in practice SA can start around 20000 degrees). An exponential fall off puts the probability near zero as the temperature lowers for costly moves and it makes drastic changes far less likely to occur. In this late stage, components are desired to settle into place, removing slight overlaps.

2.2.3 Cooling schedule

The choice of temperatures and duration of testing at a temperature is referred to as the cooling schedule. By creating a too rigorous (quick) cooling schedule, local minima are typically found instead of a satisfactory result [7]. Simulated Annealing has the property that as the timing schedule lengthens, the probability that the global minimum will be found approaches one [6]. The temperature starts at a high value and is lowered with a fractional value, α.

$$T_{new} = \alpha \cdot T_{old} \tag{2.5}$$

In Equation 2.5 the application of the α value is shown. The actual numerical value of α does not need to be constant across the SA application. A typical implementation uses two α values: one that allows the temperature to fall quickly and one that draws the decline of the temperature out. The smaller α value, α_1 is typically 0.95 while the larger α value, α_2 is typically 0.998.

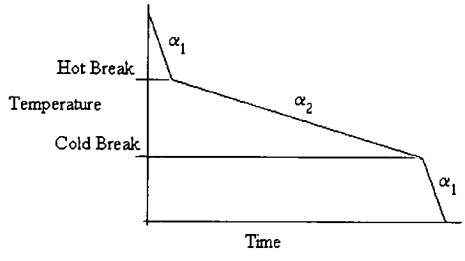

Figure 2.3: The effect of alpha parameters on temperature

The α value which decreases the temperature faster is used in the beginning and the end of the temperature range. A generalized graph of temperature versus time is given in Figure 2.3. The faster α_1 value in the upper extreme is there to make sure the design doesn't become overly cluttered with bad placements due to moves with high costs being accepted. The slower α_2 value in the lower range is there because the design will quickly settle and over iteration will simply use more run time. The middle area is where the most work is actually done. This area is bounded by the hot break and the cold break, which are temperature values that separate the different α values.

2.3 Greediness in algorithms

To have a greedy algorithm is to have an algorithm with knowledge of the underlying system. A greedy algorithm employs a heuristic to attempt to more quickly solve a problem. Greediness is potentially harmful to an algorithm since it can create local minima very easily because they typically do not operate on all the available data, but instead they home-in on an apparent solution, that is, a local minimum. For example in Figure 2.4, if the search

pattern starts at the location of the dot shown, a greedy algorithm can be expected to slide down the slope into the local minimum, and not to the global minimum on the right.

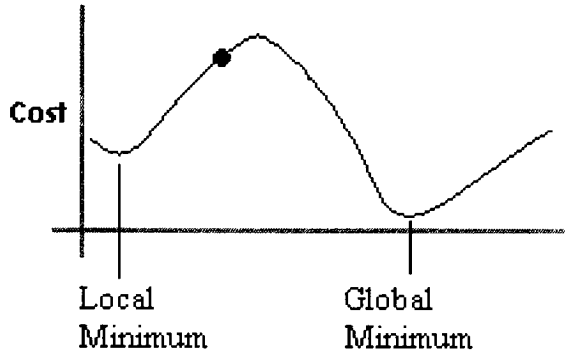

Figure 2.4: A function with multiple minima.

2.4 Force directed technique

The force directed technique has been successfully deployed where it was shown to converge more quickly to a desirable solution [9]. While using the force directed technique exclusively leads to local minima, mixing the force directed perturbations with traditional SA perturbations (non-greedy) converged quickly to a lower wire length than just SA perturbations alone. This performance provides the motivation to attempt to enhance the force directed technique as it combats Simulated Annealing's excessive iteration weakness.

The force directed technique was developed by Neil R. Quinn et al. [14]. This technique models Hooke's Law when determining placement for elements on a semiconductor die. For a spring, Hooke's law states that the force between two objects attached by a spring is proportional to their distance from the spring's equilibrium point. This proportion is given in Equation 2.6, where x is the distance that the spring's state is from equilibrium and k is a constant for the spring dictating its strength. This spring attraction can be directly related to other forms of attraction, including that created by nets in a circuit.

$$F = -kx \tag{2.6}$$

In Figure 2.5 the elements are attracted to each other because they share nets. In almost any design, every component is attracted (through other components) to every other component. Without further work, this technique would simply move every component directly on top of each other so the springs would contain no potential energy. One proposed method to solve this is to create repulsive forces between circuit-elements that are not attracted together directly [14]. When modeling the design with springs, the strength of springs (the k constant) may be defined to allow greater control of the model. This strength may be taken as the same for every net, or net connectivity could be weighted by differing the strength for different nets. In Figure 2.5, there is no direct reason for a strong attraction between the AND gate and the buffer as long as they are both not far from the source of the net. The AND gate and the multiplexer on the other hand have likely chance of being closely tied because it is generally desired that the select line to a multiplexer arrives as quickly as possible. Information about the priority of signals is not a part of the industry standard DEF/LEF format (described in secion x.x), so an alternative medium would have to be developed to convey this to the tool.

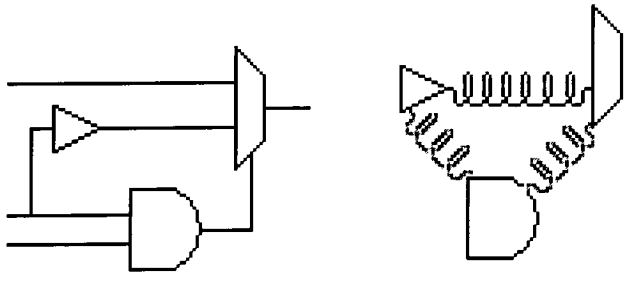

Figure 2.5: A spring representation of a circuit's internal stresses

$$Posx, y(i) = \frac{\sum_n PosX, Y(Input(n)) + PosX, Y(Output(n))}{\sum_n Count(Input(n)) + Count(Output(n))} \tag{2.7}$$

The use of division by arbitrary numbers is a quality of the force directed heuristic which makes its iterations slower than that of a standard non-greedy iteration. When the average location a component is being pulled to by a net is evaluated, division by the number of member nets in which a component resides is required. This division is shown in Equation 2.7. This equation gives the zero-force position of the current component, that is, the position where all spring forces of all nets cancel each other out. This is done by averaging the locations of all the input $(PosX, Y(Input(n)))$ and output $(PosX, Y(Output(n)))$ pins in all the nets that the component is a part of. The expectation is that the perturbations generated are better and will be accepted more frequently than without the technique.

Traditional SA perturbations are based solely on the temperature and the location of the component that is going to be moved. For single-component perturbations, a bounding box is made around the component that is the target component. Under a force directed single component perturbation, the bounding box encapsulates the zero-force position.

2.5 Min-cut

It is advantageous in placement to group circuit elements to separate logical groups by connectedness. Groups of elements which intra-communicate heavily need to be physically close or else their distance will greatly increase wire length. The force directed technique alone does not specifically recognize inter-connected logical groups, although it is somewhat addressed by net pulls as groups will tend to share nets. It is possible that two elements which are not directly connected would benefit from gravitating towards each other directly, such as in a bus. The min-cut is a technique which may be utilized to find these groupings.

Min-cut is a concept from graph theory which is defined as a cut in a graph of least weight where a graph is a collection of undirected, weighted links between nodes and a cut is a non-trivial line which separates the graph into two distinct sections [21]. This very well fits the node-net relationship that exists in the placement problem, except for the

weight quality which is free to be used as it is seen fit. The purpose of applying the min-cut to the placement problem is to find groups of components that may not have been specified, or perhaps even recognized, by the designer.

Min-cuts are not trivial to find. It is important to take into consideration the time it takes to perform a min-cut when determining if it is helpful to the Simulated Annealing problem. In this thesis the focus is directed to the SA result itself with feasibility (as determined by the time to complete a min-cut) separate. The min-cut algorithm used is simple and claims to have comparable results to those of more complex algorithms [16]. The algorithm is iterated n times, where n is the number of nodes in the system, and each run takes n steps. Before examining the algorithm, the basic step of joining two nodes together needs to be examined.

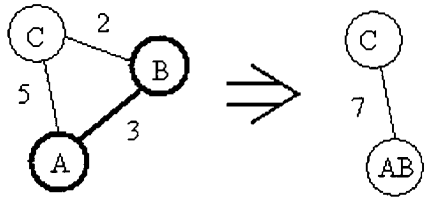

Figure 2.6: Demonstration of node combining

In the example shown in Figure 2.6, nodes A and B are chosen to be joined. Weights from common links between nodes are summed. In this example, the weights of A to C of 5 and B to C of 2 are added to make the new edge AB to C have a weight of 7, while the edge A to B of weight 3 is gone.

In the algorithm selected for min-cut V is defined as the set of all nodes. The algorithm is run once for each node in the design. The node selected becomes the starting node, a. A subset, A is made from this starting node. The neighbors of A are added into A until no nodes are left. The order in which neighbors are added into A is such that the nodes with the most heavily weighted connections to A are added first [16].

In the next example (Figure 2.7), a is the selected starting node. The weights of the

connections are shown next to each edge. The numbers inside the nodes represent the order in which they are added into subset A (starting just with node a). The last two nodes to be added, 6 and 7, are combined in a manner similar to the previous example in Figure 2.6. The weight of the combined node towards the rest of the design is recorded. The process of starting from node a and adding nodes into subset A is repeated. Eventually there will be only two nodes left. At this point, the weight of all the recorded weight is observed and the smallest one is a min-cut, given the starting node a. This entire process is rerun for every possible starting node [16].

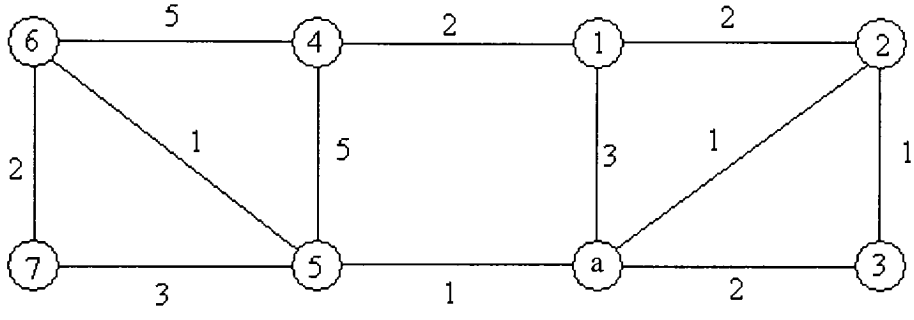

Figure 2.7: Min-cut method

2.6 Row-based design

In most modern standard cell designs, the position of circuit elements is not completely arbitrary. The vertical axis is broken up into rows which set fixed coordinates that a component may appear (on that axis). The advantages to this appraoch are significant. Power and ground routing becomes completely trivial because specific metal tracks can be reserved for that purpose. Clock routing may also be simplified. Algorithms are able to act faster on row-based designs because it is a simplification of one of the degrees of freedom. If one component overlaps another, it is simple to see if there is enough room for either component to be nudged out of the way. It also allows designs to be very densely packed because rows can be mirrored which lets them share their P and N regions, which would

otherwise require spacing (as a fabrication requirement).

There is a motivation for row-based placement from the perspective of placement software. Firstly it organizes components by dividing them into groups with a medium-sized granularity (depending on the dimensions of the design). For example a design with ten thousand components divided into a hundred rows divides the components into groups of roughly a hundred instances each. Secondly it potentially simplifies the process of determining overlap because the cells which may overlap are now better defined.

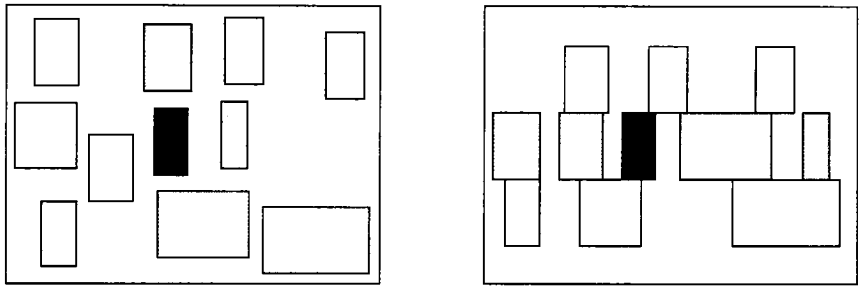

Figure 2.8: Overlap detection regions

In Figure 2.8, views from a couple designs are shown. The view on the left is from a design with no rows, and on the right is from a design with rows. The component colored black represents the focus of the overlap detection. It is desired to examine these components and determine what overlap, if any they have with other components. In the row based design, the components with which must be compared are colored gray. These are the components in the same row. When the rows are stored such that easy access to neighboring components (up to two links), it is obviously a trivial task to check for overlap on a single dimension. A method has been developed (see section 3.2.2) that allows an entire row to be quickly evaluated for overlap. Each row can be evaluated in sequence (or implemented in parallel) to quickly evaluate the overlap for the entire design.

When detecting the overlap for a component in a design with no rows, it is less clear

which components should be compared. One obvious method is to compare each cell to every cell in the design. This method could be parallelized in hardware design [2], however it is still not an efficient solution. Another method in Figure 2.8 is the bin method. In this method, the design is broken up into horizontal and vertical bins for the purposes of specifying a general location of a component. Using this technique, only components in the same bin need to be compared for overlap. It is necessary then to maintain a list of bins which contain lists of components which reside within, in a manner similar to the row-based design. This is still inferior to the row-based design because this method requires hit detection in two dimensions.

A downside to using row-based designs is that some amount of area is inevitably wasted in the process. If every component is mandated to be the same height, then many components will have to be larger than required because they cannot be shrunk vertically. This also means that a group of very small cells cannot be packed close together vertically (such as drivers for a bus), however the ease of routing control in a row based design should offset this by allowing many tightly packed drivers to hit horizontally in a row sharing horizontal metal space.

2.7 Enhancement limitations

Any optimization made to an algorithm, while it may improve the performance in regard to one problem, can decrease the performance in regard to others. There has been work done on No Free Lunch Theorems [22]. According to these theorems, the average performance of any algorithm across all problems is the same. As an example the brute force approach, notoriously known for being impossibly slow for placement applications, is the fastest algorithm for the subset of placement applications where the initial solution is at the global solution (given that the algorithm realizes this). Compare this to Simulated Annealing, which would completely disorganize the result when it starts at a high temperature and accepts all sorts of damaging perturbations. The improvements made to an algorithm must

therefore be specific to the problem at hand, with the consequences in mind.

2.8 LEF/DEF file specification

The Library Exchange Format and Design Exchange Format (LEF and DEF) describe a component library and individual components in human-readable ASCII text files. These can be used on both the input and the output of a placement tool because the specification has the ability to keep track of component locations. This specification is owned and maintained by the OpenEDA community organization [4].

2.8.1 Library Exchange Format

The Library Exchange Format file contains library information. It may be shared among designs which use the same standard cells and fabrication process. The file gives logical names to poly layers, wire layers, vias, and macro-boxes (e.g. standard-cells), among other objects. Metal spacings and widths are given for the metal layers, and sizes are given for vias, including metal size requirements on either end. Most importantly to placement, dimensions are given for each cell. Other information about the cells includes pin locations, symmetry, and an origin (for rotation). By the LEF specification, distances listed are always given in microns; however its precision can be controlled. This allows measurements of far smaller sizes than a micron.

```
MACRO MAS4
   CLASS CORE ;
   SIZE 0.04 BY 0.16 ;
   ORIGIN 0 0 ;
   SYMMETRY X ;
   SITE core 0 0 N DO 1 BY 1 STEP 0 0 ;
   PIN P1
   DIRECTION INOUT ;
```

```
   USE SIGNAL ;
      PORT
         LAYER metal1 ;
            RECT -0.0002 -0.0008 0.0002 0.0008 ;
      END
   END P1
END MAS4
```

This LEF file code excerpt defines the macro box "MAS4". Any number of components may be described as type "MAS4" and they will share the characteristics defined here. The size, 0.04 by 0.16, describes the horizontal and vertical lengths respectively. If this is a row based design, it may be assumed that all macro cells show the same height as 0.16. The UNITS statement appeared previously stating the number precision to be 1/100th of a micron, which justifies the cell size given.

2.8.2 Design Exchange Format

The Design Exchange Format file describes a design by defining its properties. The units in a DEF file are defined as relative to the units in the LEF file that the design uses. A relative unit scaling factor is given which allows conversion to whatever units the corresponding LEF file is in. This allows process changes to only affect the LEF file (if the process change is robust). Rows or other placement areas (and placement blocks) are given with coordinates. Metal tracks can be reserved for power, ground and shield routing so that routing programs will not overrun necessary space. Every component is defined, perhaps placed, as a reference to a LEF component definition. Components may also be described as fixed, not allowing the component to be moved by the placement tool. Nets are defined by a net name and a list of components and pins.

```
COMPONENTS 12028 ;
- a0 mod2 + FIXED ( 451 3280 ) N ;
- a1 mod0 ;
```

```
- a2 mod1 ;
- a3 mod0 ;
- a4 mod4 + PLACED ( -23 3113 ) N ;
...
END COMPONENTS
```

This DEF code specifies that there are 12028 components in the design. Each component is listed by a unique name, the component type (as defined in the LEF file) and a placement status. Components a1, a2, and a3 are not placed. The a0 and a4 components are placed at given coordinates with a northern orientation (usually default). A placement program would be unable to move component a0, however it may choose to move a4 depending on implementation.

```
NETS 11753 ;
 net00096 ( a1271 P0 ) ( a8496  P1 ) ;
- net00097 ( a1285 P0 ) ( a1401 P0 ) ( a31 P0 ) ( a6030 P0 )
     ( a7905 P0 ) ( a8561 P0 ) ( a9796 P0 ) ;
...
END NETS
```

This excerpt shows that 11753 nets have been defined. Two nets are shown here: net00096 connecting two components and net00097 connecting seven. Pins are listed for the connections which can be used to determine exact routing distances (by a routing utility). In the DEF specification there is a UNITS command as in the LEF specification, however in a DEF file it is a scaling factor. The typical ratio is 100:1 where the units in the DEF specification are hundredths of microns, while they are microns in the LEF specification. For example if the die area of the design is specified with the command: DIEAERA (-38346 -38360) (38412 38416), the die area is actually 767.58 x 767.76 microns.

```
ROW ROW_131 core -33330 32816 N DO 1011 BY 1 STEP 66 504 ;
ROW ROW_130 core -33330 32312 N DO 1011 BY 1 STEP 66 504 ;
```

The ROW command defines the usable area of the design (if rows are to be used). The origin for the first row is specified as (-33330, 32816) and the orientation is N (north), meaning that the coordinate specifies the upper left hand corner of the row. It is possible to create a horizontal step pattern to create slots for cells with this command. It is also possible to take the final value 504, the row height, and create data structures to contain the components.

2.9 Perturbations

There are typically two or three different kinds of cell moves (or perturbations) used. The first is a single cell move. One cell is taken at random, a new spot is chosen at random (the distance from the starting cell determined by the current temperature) and a move is attempted. If the move results in a higher cost, then it must satisfy the Boltzmann test to be accepted. This kind of move often results in an overlap. The second kind of perturbation is a swap. Two cells are selected at random and their locations are swapped. If the cells are of different widths then they will take up different areas when swapped, possibly resulting in new overlap on one side of swap, which again would result in the Boltzmann test. The final perturbation is an x mirror. Here a cell is simply mirrored along a vertical line marking the center of the design. This is not typically used any more but was used with the classic Timberwolf [15] Simulated Annealing application, which forms a benchmark for other algorithms (including this one) to compare against.

Chapter 3

Software Implementation

3.1 Framework

The software implementation is based on a previous framework [2], however it has been extended for the purpose of this thesis in a few different manners. The placement method is now row based to match common standard cell layout tools. The reason was to reduce the overlap computation time with the simplification this brings. The force directed technique has been implemented for single move perturbations and swaps. The goal of this addition was to measure the improvement with respect to iteration count and solution quality, and to provide a basis for possible speedup to be realized with a hardware implementation of the force directed technique.

A min-cut algorithm was made available for use which attempts to discover the groupings of circuit components in an attempt to make the Simulated Annealing algorithm converge faster without sacrificing the quality of the result, or achieve a higher quality result without sacrificing speed. One last enhancement introduced was a dynamic cooling schedule. Using this technique, the number of iterations the algorithm stays at a given temperature depends on the quality of the perturbations being produced. This is as opposed to the original fixed schedule of one-thousand iterations per temperature gradient.

3.2 Implementation details

3.2.1 Row-based direction

A few possible techniques of data storage were investigated for implementing a row-based placement. The framework (which was not row-based) had all components in a single array with location properties. This technique would work for a row-based implementation by changing the vertical axis property to a row number property; however one goal of the row-based implementation was to improve the time it takes to do overlap calculations over the previous implementation. To do this would require that rows are grouped such that components need only look at their neighbors in a row to detect overlap. An ordered list implementation would allow this.

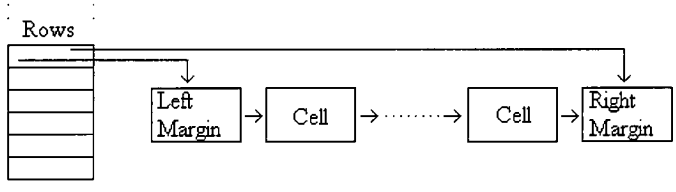

Figure 3.1: Row implementation

To allow ordering of components and to ease row iteration a linked list of components was determined to be implementable, and was constructed as shown in Figure 3.1. Since the number of rows in a design is constant and being able to address any one of them easily is a requirement, a simple array was used. Margin cells were added to the left and right side of each row (the east and west borders of the design) to act as a control mechanism and as a tool to allow rows to be empty. These cells cannot be moved and are not written to the output file; however they do have real space and are counted for overlapping concerns. They exist outside of usable area so they do not affect the overall utilization.

3.2.2 Overlap detection

One of the main goals of the row-based implementation was to reduce the overlap detection time which was previously shown to be significant [2]. This provided the motivation for the linked list implementation. Previously the overlap was determined and given to an individual cell as its overlap. In this implementation, the overlap resides completely inside each row (since two components in different rows cannot overlap) so therefore the overlap is given to the row, not the cell. The overlap cost calculation for the row is a simple doubly-nested loop with a worst case timing on the order of *O(n log n)*, where n is the number of components in the row, if every component in the row overlaps with one another. The best case timing is order *O(n)*, when there is no overlap. If a component is perturbed inside its own row or swapped with a component inside its own row then overlap only needs to be evaluated once for a single row. If a component is swapped or perturbed to another row, overlap needs to be evaluated for both rows. Below the overlap calculation algorithm is given.

1. Select the first component in the row as the primary component.
2. Select the next component in the row as the secondary.
3. If there is overlap between the two, record it. If this is not the last comparison, step the secondary component down the row and repeat step 3.
4. If the primary component is the second to last cell then the algorithm is over.
5. Otherwise, increment the primary component down the row, and reset the secondary component to the one after the primary, repeat step 3.

The detection is fast because the list is sorted by the left-most edge of each component. This adds some complexity to the process of adding cells to a row (order n at the worst). To detect overlap, the first cell in the row is selected and its right border is recorded. If the second cell has a left border left of this right border, then the overlap counter is accumulated by the difference. Since this is a sorted linked list, if there is no overlap between this

first component and this second component, then there is no overlap between the first component and any other in the design. If there is overlap between the first and second, then the first and third must be checked, and if overlap exists there then the first and fourth must be checked, and so on. Once the first component shows no overlap, the second component is checked versus the third component, and so on.

3.2.3 Overlap correction

When a Simulated Annealing run hits its shut-off temperature, ordinarily execution stops there regardless of the legality of the placement. This is not wise because invalidates the final result, perhaps unnecessarily. There is the possibility that trivial overlap remains which can be easily fixed by nudging a few components to one side or another. A simple row-based overlap correction scheme has been developed. Once SA has completed, each row is checked for overlap. If overlap exists, then the utilization of the row is calculated. If the utilization of the row is less than 100%, then it can be possible to legalize the row simply by sliding cells back and forth, without needing changing their order. This is accomplished in a number of runs and does have a potentially long runtime. It has been implemented so that the final results of the SA runs are valid, legal placements which can then be compared on fair ground to other tools.

3.3 Force directed technique

The force directed technique needed to be implemented in two separate places. Single moves and swaps are generated by different mechanisms inside the SA application. The implementation for both cases was based upon keeping a lot of randomness in the perturbations since previous work indicated that behavior which is too forced puts the design into a local minimum [9].

3.3.1 Single moves

A non-forced single move perturbation is restricted by the boundaries of the design and by the current temperature of the system. The temperature of the system provides a restriction parameter which is multiplied by the height and width of the design. The restriction parameter is determined as shown in Equation 3.1. The hotbreak temperature of the system is the point when the smaller α_1 is no longer applied and the larger α_2 is used. Until the hotbreak temperature is reached, the restriction is more than one, which means that a component can be perturbed to any location in the design.

$$Restriction = \frac{CurrentTemperature - FinalTemperature}{Hotbreak - FinalTemperature} \tag{3.1}$$

When the restriction is multiplied by the height and width of the design, these values give the vertical range and the horizontal range respectively. An example with a restriction around 0.3 is shown in Figure 3.2 on the left. A bounding box is made around the component that is being perturbed such that the edges (top, bottom, left, and right) are located one range away from the component in each direction. These are then cropped to the size of the design to avoid moving a component outside of the boundaries (in this figure the top border has been cropped). A random location is then selected inside these boundaries and the move is evaluated.

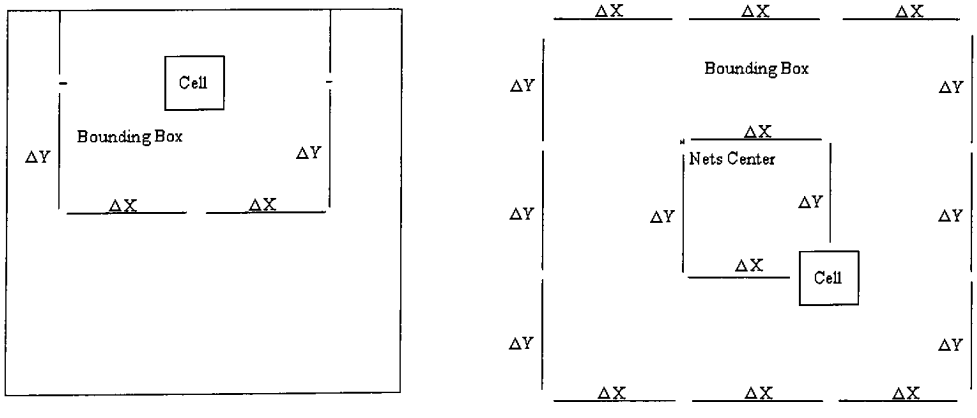

Figure 3.2: Bounding boxes of a standard and forced perturbation

In the force directed implementation, the desire was to have random behavior lie in a non-greedy perturbation to help avoid local minima. The bounding box was moved from the center being around the component being moved, to the halfway point between the component and the center of its member nets. The bounding box itself is potentially much smaller in size. The starting width and length is the distance of the component from the average location of nets. The box is constructed with this same distance around both the component and the average location of nets. This bounding box (shown in Figure 3.2 on the right) is then applied a restriction which narrows it by a fraction. Unlike standard single perturbations, this restriction is capped to a minimum of 0.2 to allow more mobility at low temperatures. This should not cause a problem given the probabilistic evaluation.

This implementation does require more calculations than a standard non-force perturbation. The average locations of nets needs to be evaluated which means that division must be done. Two methods were explored for determining the average location of nets. The first was to simply average all the mean net locations. This average is arguably correct; however it ignores the fact that some nets are larger than others. This may be the intention, however if a component has a net with two components and a net with fifty components, the later might be more important than the former. The other method is to add the sums of coordinates of all the nets and to average them at the end. This actually removes the concept of nets entirely because it's simply the average location of all connected components. The second method is more true to the force directed technique and it demonstrated better results, so it is in the final implemented method.

3.3.2 Swaps

A non-greedy (force directed) swap is a trivial matter compared to a force directed swap. The only operation is to pick two components to swap. The first component involve in the swap is randomly picked in either a traditional SA swap or a force directed swap. In a traditional SA swap the second component as also picked completely randomly. The randomness is very simple however it does violate the concept of temperature somewhat:

moves across long distances are supposed to wane as the temperature decreases. Still, these swaps prove valuable in execution and they are still somewhat restricted by the temperature since moves which raise the cost of the system will be accepted less at lower temperatures by the Boltzmann ratio.

In a force directed swap, the average location of the nets of the first component is calculated in the same manner as in the force directed single move perturbation. The bounding box is constructed, just like in the forced directed single move. The size of this bounding box is affected by the temperature so there is a greater restriction on this move than the non-greedy swap. Once the target area has been mapped, a random location is chosen inside this box. This location contains a row and a horizontal position. The row is iterated down until a component is found at the general location pointed to by the random number. This becomes the target and a swap takes place. There is always the chance that the component found is in fact the original component initiating the swap, or that there could be no components in the area. If the operation is unable to find a component with which to swap, then a new first component is randomly selected. Otherwise, an infinite loop would occur if the first component that was selected had no possible swapping partner.

A poor acceptance rate of the swapping perturbation in early tests prompted work to develop a swap that would be accepted more easily while not disrupting the forced-directed nature or significantly altering run time. The use of rows already required that the row be traversed to find a target component for a swap. Since this time is already being taken, it was decided to score each component in the row for a measure of how good of a swapping candidate it would be. The lowest scoring candidate cell would be selected and the result of the swap would be scored and the algorithm would continue from there. The score for each candidate is determined by two factors: the distance of the cell from the selected target location, and the difference in horizontal size from the originating cell. This size difference is important because the larger it is, the greater the chance of an overlap. This is mitigated by assuming that any difference in horizontal size will result in overlap, so the difference is multiplied by the overlap factor set in the SA application. The added complexity was

rationalized by a very large gain in the percentage of accepted swaps.

3.4 Min-cut

Before a min-cut can be done on the design, the way the design is described must be changed. Originally for the purpose of the Simulated Annealing application, nets were described as objects between several components. This is partially due to the way that nets are stored in a DEF file, where a net name is given and then the associated components are listed. It also allows the algorithm to easily use the size of a net, or easily iterate through the components of a net. For a min-cut to be performed, a mathematical graph must be constructed where nodes are joined by edges.

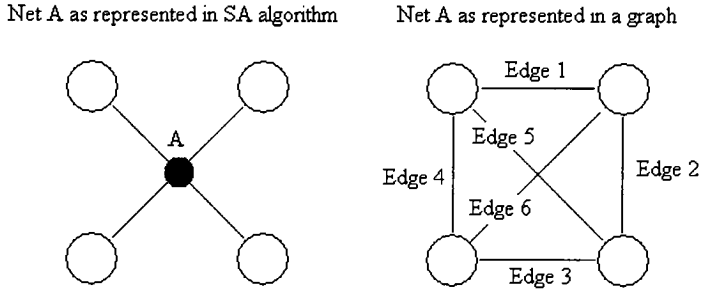

Figure 3.3: Net versus graph representations

As shown in figure 3.3, the net representation (left) is a single object which lists all connected components. In the graph representation (right), each node is individually connected to each other with a bidirectional edge. The operation of creating the edges and nodes of the graph from the original is not trivial and takes time on the order of $O(n*d\hat{3})$, where n is the number of components in the design and d is the degree of the graph. Note that it is the absolute worst case because not all nodes will have d edges.

The min-cut algorithm itself takes n (where n is the number of nodes in the design) iterations and is listed in pseudo-code below. Each iteration performs a min-cut starting with a different node as the seed node. It starts by copying the graph version of the design

which is on the order of $O(n*d\hat{2})$. Once again this is a worst case assessment. Inside each iteration there are two nested loops. The outer loop checks to see if the current min-cut iteration is completed. This will occur if there are only two nodes left (see section 2.5), if an abort has been called because no satisfactory min-cut can exist, or if a best-case min-cut has been found. A best-case min-cut would be two groups of nodes of equal size with one or zero connections between them. There is a possibility that the nodes that are to be cut have no satisfactory min-cut if they are limited interconnectivity. For example a group of fifty nodes where each node is connected to only one other node, there is no motivation to perform a min-cut and doing so would increase the complexity of the algorithm. This outer loop takes n iterations at worse.

```
1. for( n from 0 to number of components){
2.    subset A = node n
3.    while(nodes left is more than 2){
4.       while(nodes outside A is more than 2){
5.          add most weighted connection to A
6.          exit if best-case min-cut exists
7.          exit if min-cut isn't desirable
8.       }
9.       merge remaining two nodes
10.   }
11. }
```

The inner loop performs the min-cut as described in (see chapter 2, section 2.5). It is executed *n* times on its first and decrements in count by one for each iteration until finally just once on its last run inside the outer loop. Again there are possibilities for early escape from this loop if a best-case min-cut is found or if a min-cut is determined to not be worth finding for the given node set. In each iteration of the inner loop the solution is scored according to the connection weight between the nodes inside group A and outside, and by their relative sizes. A min-cut where one node is separated from the others with a

connection weight of one is not a valid min-cut, so difference between the number of nodes in the two groups is taken, multiplied by a constant, and added to the weight between the groups to score it.

There is further complexity involved which is not apparent that slows down the min-cut further. For example merging two nodes requires a doubly nested loop which fixes back references in edges. A pair of loops is also required to add a new component into subset A to update edges and a quick sort routine is used to order node links from subset A (which avoids the use of another doubly nested loop) to combine multiple links to the same nodes.

The min-cut depth is specifiable to a power-of-two number of groups. This is performed by successively running the min-cut on the results of previous min-cuts. The desirable number of groups may not be reached because a group created by a min-cut may be loosely connected and therefore not produce a meaningful min-cut.

3.5 Cooling schedule

The challenge with a cooling schedule is that if the temperature declines too quickly, the solution may converge to an undesirable local minimum [11]. If it is too slow, there will be wasted cycles of relative inactivity as perturbations will be thrown out in large numbers. The middle ground is desirable and can be approached in a couple different ways. In the implementation (described in section 2.2.1), the temperature decreases from twenty thousand degrees to one degree (abstract units), with an α_1 of .95, an α_2 of .998, a hot break at thousand degrees and a cool break at five degrees. These numbers are kept for comparison purposes. The number of iterations at each temperature is set at one thousand. This process results in 2,724,000 iterations being run for any given design on input. The intention is that the user determines the best α values, hot/cool break values, and number of iterations per temperature value.

An adaptive (or dynamic) cooling schedule has been implemented in attempt to simplify this process and to increase the robustness of the algorithm. Instead of a fixed number of

iterations being performed per temperature, the temperature can be decremented early if no successful perturbations are being made. A successful perturbation is one which lowers the cost (as opposed to an accepted perturbation, which is randomly accepted but raises the cost). The parameter of the number of iterations to perform at each temperature is now the maximum number of iterations to perform and will only be reached with many successful moves. This allows the maximum number of perturbations per temperature gradient to be made higher without significantly slowing down the application because if extra iterations aren't needed, they won't be done.

The smarter cooling schedule is implemented in the following way: firstly at the highest temperatures (before the hot break) the success rate of perturbations is very high so there is no need to assess the quality of perturbations. The number of perturbations being performed at the very high success rate doesn't need to be too large because many bad moves are accepted probabilistically at this stage. Therefore the maximum number of iterations for these temperatures has been decreased. The remaining temperature values start off by allowing one hundred perturbations. From then on if the percent of successful perturbations is less than 2%, the temperature passes to the next value. The temperature also passes to the next value if it hits the maximum iterations per temperature as specified for run.

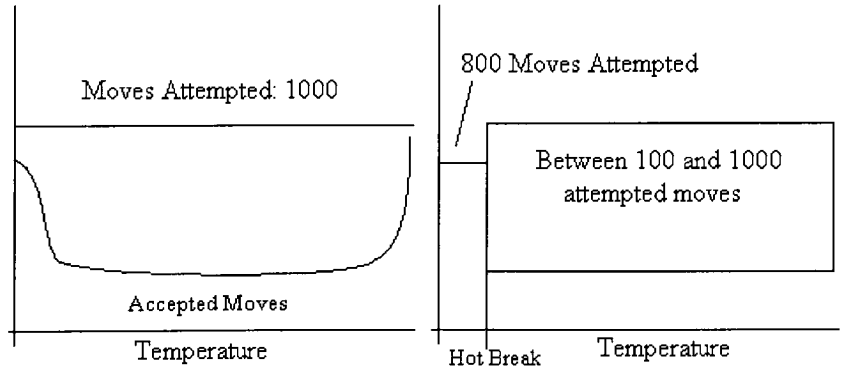

Figure 3.4: A constant cooling schedule versus a variable one

In Figure 3.4 two different cooling schedules are compared. The first is the previously implemented constant cooling schedule allowing 1000 iterations per temperature value.

Many moves are accepted in the beginning and the end because the beginning moves are accepted probabilistically at the high temperature and the ending moves are accepted because they are small movements designed to fix overlap. The second graph contains a variable cooling schedule where the number of iterations is based on the number of accepted moves at a given temperature. While less iteration will be conducted at every temperature level, it is the intent that a greater portion of all iterations tried will be accepted since the low end of the performance is capped at 2%.

3.6 Results

3.6.1 Benchmark selection

Benchmarking was required for comparing the SA results with those of other tools, and for evaluating how changes affected the application. The IBM-PLACE [1] suite was selected for cross-application comparisons because of its longevity and inter-operability among placement programs. The suite contains large circuits starting at twelve thousand cells it increases from there. The original framework for this SA application was benchmarked with this suite; however the version was a little different and it was not row-based. In the IBM-PLACE benchmarks, the first two numbers refer to the circuit number and the last two numbers give the utilization of the design [1]. Each design has an easy and a hard implementation, where the hard implementation has a higher utilization.

The ISCAS [10] suite was also selected for use. This suite provides designs with lower cell counts ranging from 160 to 3500. This proved ideal for testing the computationally heavy min-cut algorithm.

3.6.2 Row-based implementation

It was believed that utilizing a row-based implementation would cut the amount of time spent diagnosing overlap. In the previous work [2] the amount of time spent on the delta

cost calculation was around 75% to 85%. With the use of the row-based overlap detection this time has fallen to 55% on average, with the percentages for some of the IBM-PLACE and ISCAS benchmark circuits given in Table 3.1.

Bench	Cell Count	Cost Time (Total)	Time (Overlap)	Time (Wire length)
c432	160	49.0%	40.7%	8.3%
c499	202	50.1%	40.0%	10.2%
c880	383	51.5%	44.4%	7.1%
c1355	546	51.6%	44.2%	7.4%
c1908	880	54.3%	38.4%	15.9%
c2670	1269	64.4%	52.0%	12.4%
ibm01-85	12028	71.2%	59.7%	11.5%
ibm02-90	19062	74.2%	63.4%	10.8%

Table 3.1: Portion of SA runtime devoted to cost tasks

The decrease in overlap detection time may also be explained by the additional overhead associated with a row-based system. While much less time is spent on detecting overlap and cost in general, more time is spent on moving components because the linked-lists which make up the rows need to be iterated through. It is expected that a taller design would be placed more efficiently than a wider design because linked list would not need to be iterated through as deeply. A trend in an increasing amount of time being spent on overlap detection with the increase in design size is seen.

3.6.3 Min-cut

The min-cut did not drastically improve the results of the Simulated Annealing application. In Table 3.2, the results of the min-cut are given for a few smallish circuits. It is noticeable that smallest circuits are not affected as much by the min-cut. The lackluster results are likely due to the min-cut is adding complexity to a relatively simple problem.

Larger circuits add complexity to the min-cut procedure. As the number of cells in the problem circuit increases, the running time of the min-cut algorithm becomes prohibitive. Information regarding the runtime of the min-cut algorithm is given as a fraction of the

		Wire lengths for given number of groups					
Bench	Cell #	1	2	4	8	16	Best
c432	160	494949	516447	543983	510002	518142	-4.3%
c499	202	685649	685366	690467	686635	670892	2.2%
c880	383	1169395	1268027	1161192	1205538	1181374	0.7%
c1355	546	1789855	1720634	1844729	1550390	1647855	13.4%

Table 3.2: Min-cut performance results

SA time (not including min-cut) for a typical run in Table 3.3. A typical run is that as previously described in section 2.2.1.

		Min-cut Time / SA Time			
Bench	Cell #	2	4	8	16
c432	160	0.3	0.31	0.32	0.32
c499	202	0.49	0.61	0.61	0.62
c1355	546	12.26	13.18	13.88	14.15

Table 3.3: Min-cut timing information

Since the goal of the min-cut is to create a grouping which is then used to clump components, it is desired to see if this clumping activity actually occurs. After a min-cut run is complete it is possible to write a FIG file which shows separately the min-cut groups by pattern. In Figure 3.5 the result of the ISCAS c1355 benchmark with four min-cut groups is shown with horizontal, vertical, and diagonal lines respresenting the groups. It can be seen that the min-cut has successfully grouped many cells by location, although instead of a large clump the groups seem to be divided into a couple pockets each.

3.6.4 Force directed technique

The force directed technique has been evaluated. According to prior work, the force directed technique will quickly result in a local minima if used too extensively [9]. The results gathered by this thesis appear to disagree. The technique was tested on eight differently size designs with component counts ranging from 160 to 19062 cells. Smaller designs, such as the ISCAS c499 circuit did fairly worse with more greedy perturbations

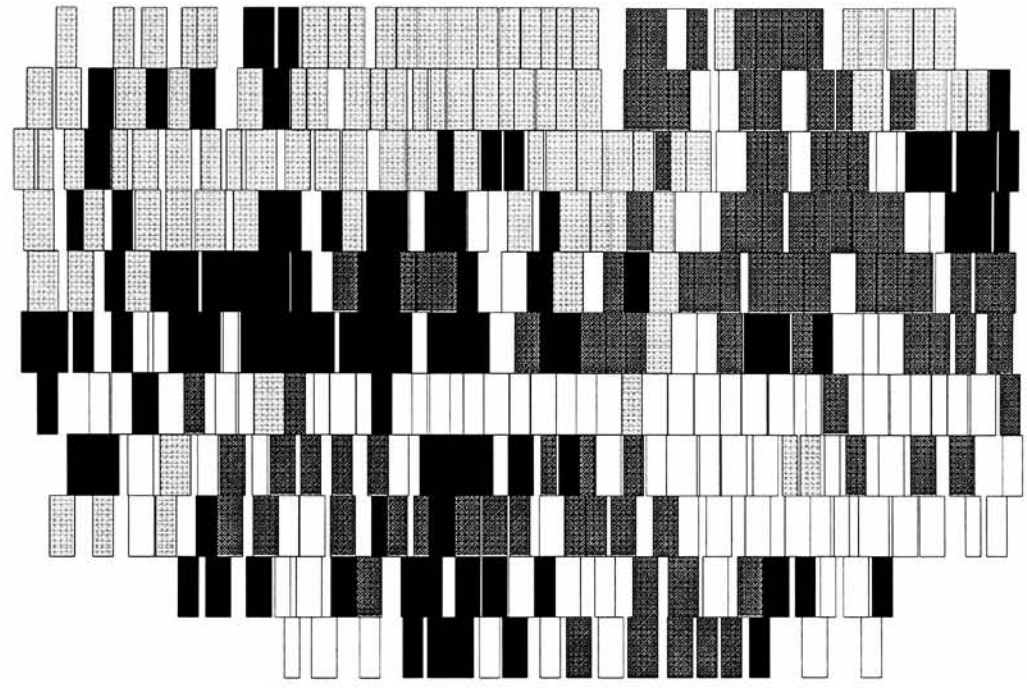

Figure 3.5: Mincut results from ISCAS c1355

as seen in Figure 3.6. In Figure 3.7 the results of a large design, ibm01 are shown. The impact of the force directed perturbations are extremely obvious in this case.

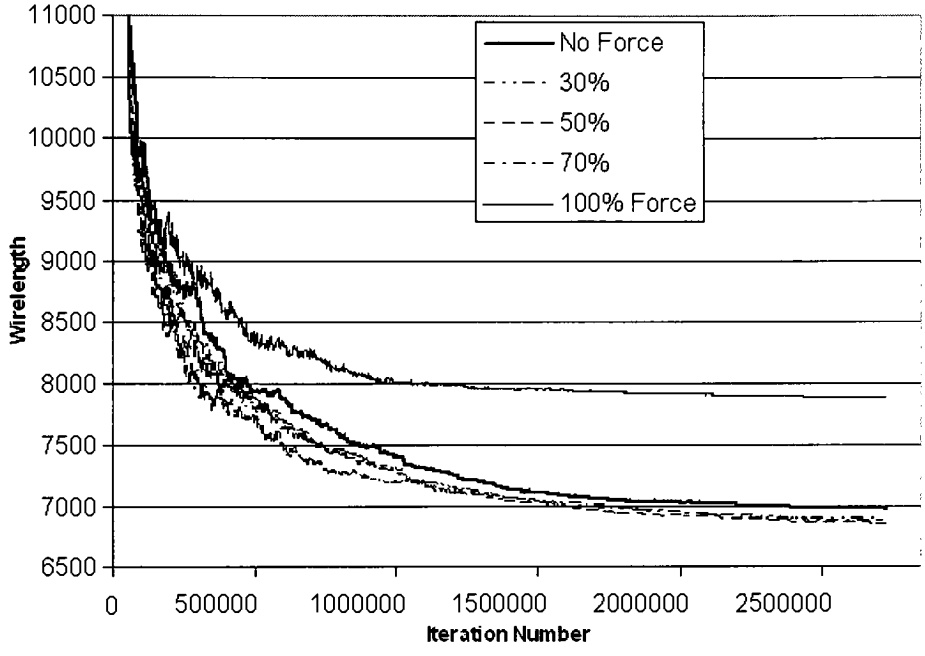

Figure 3.6: Forced results from c499 test-bench (202 cells)

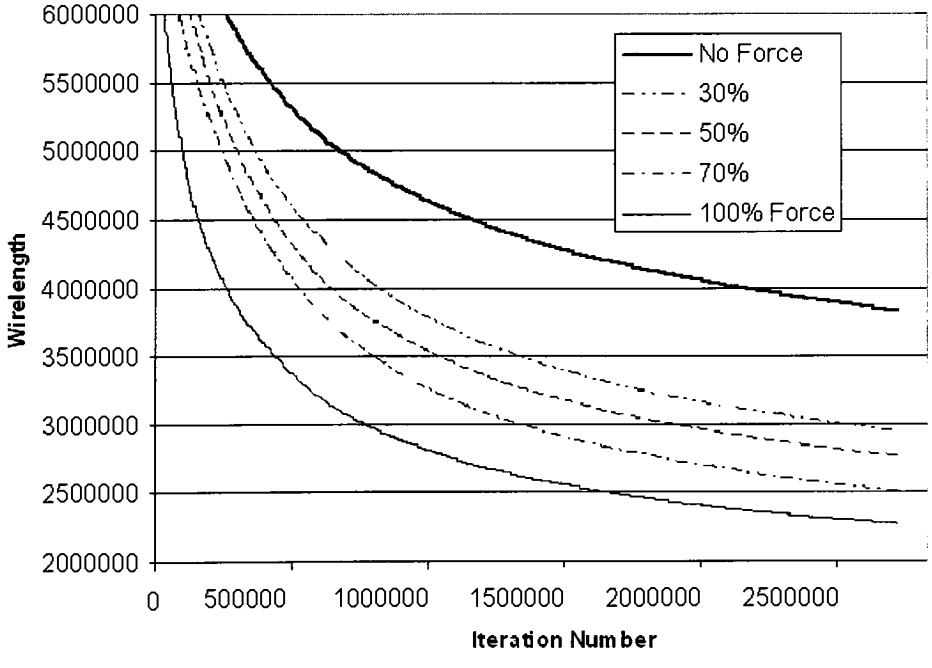

Figure 3.7: Forced results from ibm01 test-bench (12028 cells)

In the larger designs a trend developed where the more force directed moves were used, the better the results achieved for a given number of iterations. The main cause for this is that the quality of the iterations produced increased dramatically with more forced perturbations. This was shown to be especially true in the case of swaps. The enhanced swapping method created is responsible for this large number of accepted swaps. The percentages of accepted perturbations for varying degrees of forced perturbations is given in Table 3.4.

	Percent Singles/Swaps Accepted				
Benchmark	No Force	30% Force	50% Force	70% Force	All Force
c432	1.57/0.81	3.37/3.25	4.49/5.53	3.67/4.92	5.48/9.19
c499	1.60/0.77	2.77/2.37	3.35/3.87	3.92/4.98	5.07/8.60
c880	1.37/0.70	2.96/2.21	4.36/3.96	4.60/5.50	8.91/8.73
c2670	2.19/0.79	4.63/4.15	6.17/6.57	7.67/9.75	9.85/14.32
c3540	2.26/0.86	3.90/3.30	5.17/4.96	6.52/6.47	7.69/10.39
c5315	2.11/1.01	3.85/3.51	5.12/5.30	6.31/7.51	8.45/10.33
c6288	2.11/1.47	4.50/5.65	6.62/8.71	8.77/12.51	11.62/19.42
c7552	1.96/1.44	3.68/4.61	4.87/6.53	6.39/8.95	8.87/11.90
ibm01-85	2.14/2.46	2.11/5.26	2.28/6.63	2.52/7.88	3.87/8.95

Table 3.4: Perturbation acceptance percentage

3.6.5 Dynamic cooling schedule

By removing the rigidity in the number of attempted perturbations performed per iteration, similar results were obtained after less time. The results are given in Table 3.5. To explain this table, the line concerning benchmark ISCAS c432 is read as follows. The benchmark contains 160 standard cells. Using the new cooling schedule, it completes in 47.81% of the iterations that the original cooling schedule took (or 52.19% less iterations). The new cooling schedule produced a result with a wire length 6.5% longer than originally. The new cooling schedule's wire length after it completed is 2.51% more than the original cooling schedule *after the same number of iterations*. The metric is brought to attention because the results show that the faster cooling schedule produced better results in less time in all but two runs.

Benchmark	Cell #	% iterations	% Cost1	%Cost 2
c432	160	47.81	6.50	2.51
c499	202	37.44	0.05	-3.67
c880	383	58.86	3.34	-1.09
c1908	880	60.30	6.39	2.91
c2670	1269	72.23	-0.80	-2.62
c3540	1669	55.50	1.69	-2.06
c5315	2307	45.01	0.82	-4.28
c6288	2416	84.67	-2.26	-2.93
c7552	3513	54.20	1.02	-2.11

Table 3.5: Effects of a smarter cooling schedule

During the regular cooling schedule, the SA process does not have a way to figure out what the final wire length of the design might be. What the data in Table 3.5 shows is that the new cooling schedule completes the Simulated Annealing process at close approximation of the final results.

3.6.6 Performance comparison

Two tools Capo [18] and Dragon [20] were chosen to compare the SA application. The benchmark took place on a 2.2 GHz AMD/Linux machine. Three runs of each benchmark were performed and the time and scores of those runs were averaged. The runs produced similar results so it was determined that no further runs were required. The results of the tests are shown in Table 3.6. In this table the HPWL (half-perimeter wire length) is given in thousands of microns and time is given in seconds. The last field, comparison, is the average score of the SA application divided by the average score Capo and Dragon (lower is better).

The run for SA was conducted with 100% greedy perturbations as it was previously evaluated to produce the best result. The runtime was extended to fall between the runtimes of Dragon and Capo by increasing α_2 from 0.998 to 0.9995 and by increasing the number of iterations per temperature gradient from 1000 to 1700. Before this change the SA application finished in much less time with slightly worse results.

		SA		Capo		Dragon		
Bench	Cell #	Time	HPWL	Time	HPWL	Time	HPWL	Comparison
ibm01-85	12028	381	1,788	252	529	420	584	3.21
ibm01-88	12028	412	1,793	251	527	414	559	3.30
ibm02-90	19062	585	3,933	564	1,468	686	1,543	2.61
ibm02-95	19062	616	4,028	577	1,481	667	1,476	2.72
ibm07-90	44811	1077	16,272	1396	3,327	1133	3,521	4.75
ibm08-90	50672	1128	19,599	1580	3,594	2835	3,615	5.44

Table 3.6: Placement tool comparison

The SA application performed within an order of magnitude of the established tools benchmarked. While the wire length fell short of the competitors, a strong improvement is shown over the previous version of the SA application [2], as shown in Table 3.7. The significant time difference is likely from the change in the way that overlap is calculated between the tools. The results may still be useable by a routing application.

		Current SA		Previous SA	
Benchmark	Cell #	Time	HPWL	Time	HPWL
ibm01	12028	381	1,897,277	8760	2,080,000
ibm02	19062	585	4,602,725	8460	6,010,000

Table 3.7: Comparison to previous SA implementation

Chapter 4

Hardware Model

4.1 Framework

The hardware model is also based on a framework provided by prior work [2]. The original model was designed with the intention of speeding up the slowest part of the Simulated Annealing software implementation. It was discovered that the slowest part of the SA algorithm is the overlap detection functionality. This is still the case with the software implementation in this thesis. Since a row-based design has been implemented in software in part to speed up the overlap detection system, the same will be designed in hardware. This allows an approximation of the speedup given by utilizing a hardware device. Additionally the force directed perturbation heuristic has been modeled for hardware to evaluate its performance and possibility for hardware enhancement.

4.2 Overlap detection

The overlap detection logic should take full advantage of the row-based model being used. This can be done by emulating the method of overlap detection in the software. It is assumed that there is a storage array of components which can easily iterate through a row in order from left to right and retrieve position information. In the software implementation the left most component is chosen. The remaining components to the right (in the row) are iterated through checking for overlap between the first component's east side, and the

others' west sides. Once a component does not overlap the first, it is known that none of the other components overlap it because they are ordered. Once a non-overlapping second component is found, the first component is iterated. This has a worst case timing of order $nlogn$, where n is the number of components in the row, if every component overlaps, and a best case timing of order n if no components overlap. Every row is completely independent so there is no limit to the parallelization possibility (except the number of rows).

To implement this method in hardware an accumulator is required to keep a count of the overlap as the row is analyzed. Control signals need to go between the accumulating hardware and the hardware which contains data on the components so that the required information is available at all times. If possible, it is desirable that the number of clock cycles required to complete the row overlap detection is similar to the order of the problem *O(n log n)* and this can only be accomplished with a streaming data system.

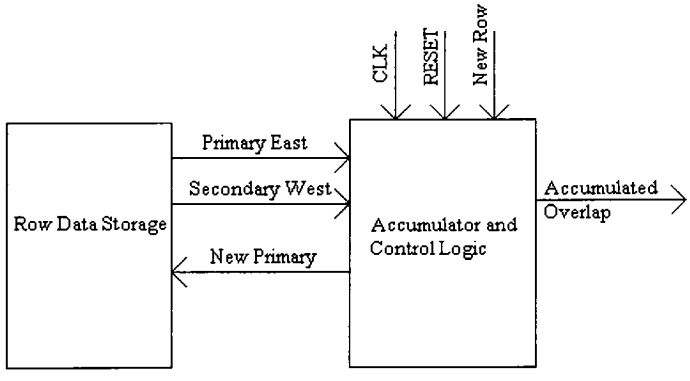

Figure 4.1: Block diagram of hardware based row overlap detection

Figure 4.1 shows the control and data signals used by the overlap system. The expected functionality of the row data storage is that with each clock cycle it will produce a successive secondary component unless the *New Primary* signal is assert, in which case it will send a new primary component and an appropriate secondary component. This signal becomes asserted when there is no overlap between the East and the West signal (as per the algorithm). The accumulation is reset on a *New Row* signal. During each cycle one

comparison is made, along with a subtraction of the East and West signal. If the comparison indicates that there is an overlap, then during the next cycle the overlap is added to the accumulated total, while at the same time the next comparison is made. This puts the hardware row-based overlap detection circuit at $O(n * \log n)$, where it takes $n+1$ cycles to make n comparisons. This leaves the row data storage with around half a cycle to return the needed information. This is enough time because there are only two possible messages that will need to be sent: the next secondary, or the next primary and the secondary that follows.

4.3 Force directed technique

4.3.1 Limiting net information

The software version of the Simulated Annealing algorithm is able to take advantage of all the information about the net connections of a particular component. It is not reasonable to do this in hardware without sacrificing considerable space or time. Many combinational and sequential logic gates only have a few pin connections. A compromise between correctness and feasibility is then reached by limiting the number of nets counted for a single component to four, for determining the force directed placement. It is assumed that the nets with the most components are the most coupled, so the nets are ranked by this measure and the four highest are used.

Benchmark	Force Directed	Estimated	Difference
c432	543566	530937	2.32%
c499	696054	715509	-2.80%
c880	1132443	1168222	-3.16%
c1355	1640099	1598407	2.54%
c1908	3271738	3468805	-6.02%
c3540	10403394	11121865	-6.91%

Table 4.1: Difference in force directed technique runs by wire length when four nets used

In Table 4.1 the results from a software simulation of this compromise are shown for a

number of circuits. This data may indicate that for larger designs, the compromise is not suitable. It should be noted that in even the worst case shown, ISCAS c3540 which has a wire length 6.91% more under this scheme, the scheme wins out over the SA algorithm without the force directed technique by placing a design with 10% less wire length.

4.3.2 Division

Since division is used by the force directed technique, it is a concern when applying a hardware model. Division can take a large amount of space on a physical design and it can also be very slow. The difficulty motivates approximating the division for both speed and area concerns. To design division approximation logic, it is appropriate to determine the divisions being done by the technique. This was done by modifying the code to keep information about every division performed.

Divisor	ibm01	ibm02
2	229352	260956
3	154174	112502
4	129452	113854
5	133678	209194
34	334	92
107	0	384
134	0	360

Table 4.2: Number of divisions performed for a run on two testbenches

The divisions performed by the forced direct heuristic were recorded and part of the data is shown above in Table 4.2. It can be seen here that a wide range of divisions take place, but the divisor does not get exceedingly large. A byte value would seem to be sufficient, which allows the possibility of division by a lookup table. The largest divisor for the ibm01 test-bench was 84, while for ibm02, the largest divisor was 134. If the division is approximated by the summation of sixteen divisions of powers of two, a very accurate approximation to the division is given.

The first ten division approximations are listed in Table 4.3. The worst division listed

Divisor	Lookup Value	Desired Fraction	Actual	Error
2	1000000000000000	0.50000	0.50000	0.00
3	0101010101010101	0.33333	0.33332	1.53E-5
4	0100000000000000	0.25000	0.25000	0.00
5	0011001100110011	0.20000	0.20000	1.52E-5
6	0010101010101010	0.16667	0.16666	6.10E-5
7	0010010010010010	0.14286	0.14285	3.05E-5
8	0010000000000000	0.12500	0.12500	0.00
9	0001110001110001	0.11111	0.11110	1.07E-4
10	0001100110011001	0.10000	0.09999	9.16E-5

Table 4.3: Division estimation chart with error

above in is division by nine, which is shown to be off by 0.01% from the actual division. The worst case division in the range of divisors two to two-hundred fifty-six is division by two-hundred forty-one, which is off by 0.34%. The acceptability of this amount of error was evaluated. This error was estimated by creating a best-fit line (Figure 4.2) of the errors and applying it in code to produce errors in the division as requested by the force directed perturbations.

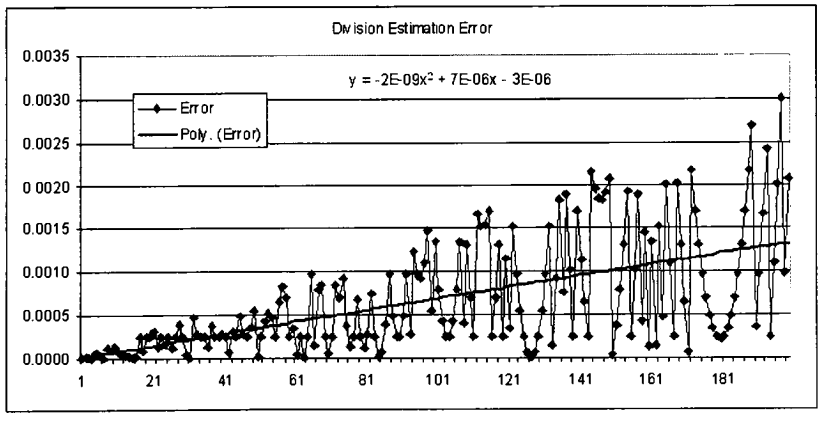

Figure 4.2: Division estimation error with best fit

The effect of this error on the final result was measured by using the division estimation of error. The result on a number of designs is shown in Table 4.4, using 70% forced perturbations. The results indicate that while the error did lower the quality of some of the results, there is no clear difference between the two. The improvements shown on some

circuits are within the possible improvement expected by varying the program's random seed.

Benchmark	Force Directed	Estimated	Difference
c432	543566	521688	4.02%
c499	696054	705699	-1.39%
c880	1132443	1124751	0.68%
c1355	1640099	1721781	-4.98%
c1908	3271738	3103363	5.14%
c3540	10403394	10436459	-0.32%

Table 4.4: Difference in force-directed technique runs when estimation is used

The cost of the force directed technique may ultimately be in the form of the utilization of hardware space. The division table as described here requires 4096 bits of ROM. This number is required for every division that is to be conducted simultaneously. A multiple-read interface could be constructed to mitigate the space requirement of multiple divisions if it becomes an issue in the design.

4.3.3 Controller logic

The functionality of the force direction perturbation module becomes apparent from the necessary limitations. The average locations of each net belonging to the component are taken as an input. This number of nets is limited to four as explained. These averages can be found in hardware using the division logic previously described, they are assumed external to this unit because it is not desirable to calculate these averages every time a component is selected for a forced perturbation. The horizontal components and the vertical components of the average net locations are separately averaged. This calculation can be accomplished in a couple cycles. During the first cycle, the parts of the first and second net, and the parts of the third and fourth net are summed. During the second cycle these sums are added and this result is sent to the division logic. Since the chance is that the division will be trivial (division by one, two or four is likely), it is only necessary to be able to divide by three with the circuit, which will save on space.

After these divisions are completed, the result is a pseudo-zero-force location. Depending on the temperature of the system and the position's proximity to design boundaries, a window will be produced in which a random number must be generated to determine a new location. This portion is not the goal of this thesis. The intent is to determine the speedup given by the force directed technique when implemented in hardware by modeling the portion of the force directed technique which is not similar to a standard perturbation.

4.4 Results

4.4.1 VHDL divider logic

The divider was modeled in VHDL with more emphasis placed on speed than accuracy, as represented by this division model. This was done with the belief that a low-accuracy division would not significantly impact the force directed technique's ability to quicken the convergence onto a solution. In the waveform shown in Figure 4.3, the value 226,497,291 is divided by 213. The result reached by the circuit available after five cycles is 1,061,014. The actual result of the division is 1,063,367.56 The error from this calculation is therefore 0.221%.

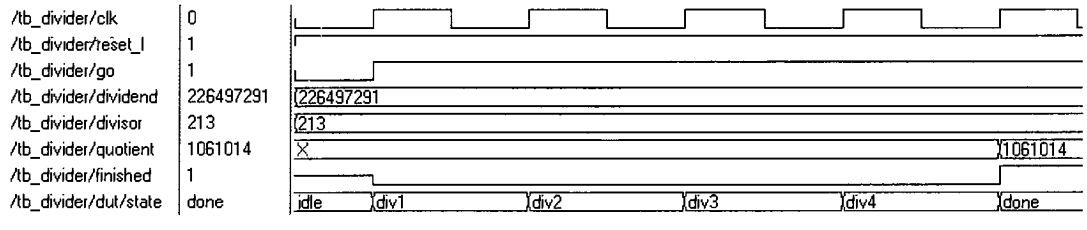

Figure 4.3: Waveform of divider logic

4.4.2 Row-based overlap detection

The overlap detection logic was designed for speed and this was achieved by evaluating one possible overlap every cycle. Needless evaluations were avoided by using an ordered

list of components.

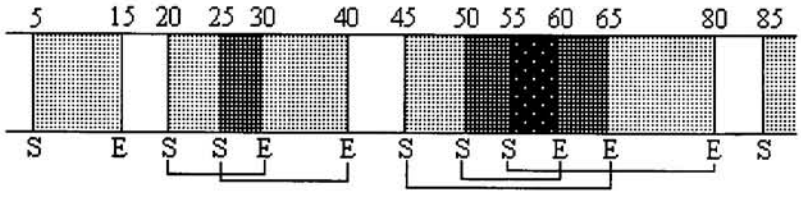

Figure 4.4: Row with overlap

A hypothetical row is shown in Figure 4.4. In this row representation the horizontal coordinate of each area of interest is given and for clarity a marker shows if a cell is starting or ending at that coordinate. The lines underneath indicate the starting and ending pairs of cells to make the cell boundaries more obvious. It should be noted that the computed overlap isn't what it appears to be. In Figure 4.5 the row is analyzed for overlap. From the row diagram it appears that the overlap is twenty-five, however the result is thirty-five. This is due to an overestimation done by the overlap algorithm that doesn't detect when one cell entirely resides inside another. It can be pointed out, however, that this circumstance should be scored more harshly in any case because it isn't as easy to alleviate with a small perturbation.

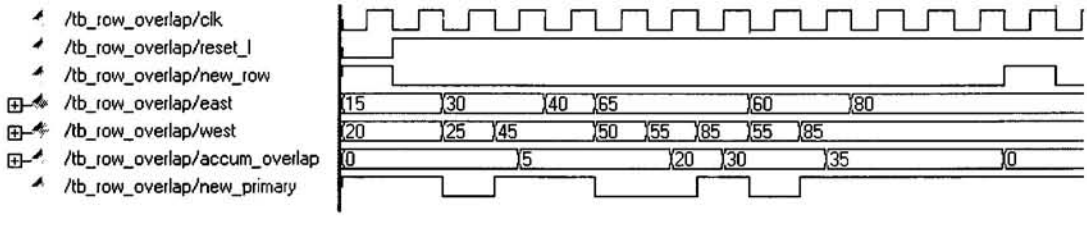

Figure 4.5: Waveform of row-based overlap detection logic

4.4.3 Speedup

A possible target system of the hardware model is the XC2VP20 Virtex-II Pro FPGA. The FPGA runs at a clock rate of just over 200 MHz. The previous hardware model speedup

has been evaluated for this board [2], so it is appropriate for this one to be as well. In Table 4.5, a comparison is given between the FPGA clock cycle requirements for overlap detection of this work and the previous work.

Action	Previous Implementation	Current Implementation
Overlap Detection	2 cycles	1 cycle*
Overlap Calculation	4 cycles	1 cycle*
Overlap Accumulation	2 cycles	1 cycle*

Table 4.5: Hardware model performance between the previous and current implementation

Overlap detection and calculation are now done in the same step and accumulation takes place during the following step if the detection came back positive. At this step, the next component is already lined up and overlap detection and calculation is being performed on it. This allows the new implementation to do a stream of overlap detections much more quickly than the previous implementation. In Table 4.6, a comparison is given between the cycle requirement of the AMD 2.2 GHz GPP and the 200 MHz FPGA processors. The number of row cost evaluations, cell overlap evaluations, and FPGA cycles required are given in thousands. The overlap times listed are in seconds.

				Overlap Time		
Benchmark	# row cost	# cell overlap	FPGA Cyc.	GPP	FPGA	Speedup
ibm01-85	19,501	2,001,346	2,020,847	215.88	10.10	21.37
ibm01-88	19,516	2,091,475	2,110,991	229.71	10.55	21.77
ibm02-90	20,026	3,247,402	3,267,428	354.84	16.34	21.71
ibm02-95	20,010	3,674,514	3,694,524	379.65	18.47	20.55
ibm07-90	20,336	5,289,083	5,309,419	704.09	26.55	26.52
ibm08-90	20,470	5,971,791	5,992,261	777.20	29.96	25.94

Table 4.6: Hardware model performance comparison to software implementation

The hardware model requires one cycle to evaluate each cell overlap, plus an additional cycle per row evaluated, so summing the two gives the cycle requirement of evaluating all the overlap. The speedup given is the time the overlap calculation takes on a general purpose processor divided by the time it is predicted to take on a 200 MHz FPGA. The average speedup provided for these test circuits is 22.98.

Chapter 5

Conclusions

The intent of this work was to enhance a previous Simulated Annealing application by adding row-based placement and by creating a min-cut procedure for the purpose of uncovering component groups. The improvements in performance in both run-time and quality of product were benchmarked against standard tools in industry. The hardware model was updated to include row-based overlap detection and greedy perturbation generation, including an approximate division circuit. The effect of the division approximation was modeled in software to validate the assumption that perfect division was unnecessary. Timings between an FPGA and the software implementation were analyzed giving the ideal speedup that could be expected.

5.1 Discussion

The force directed moves implemented in this project did not appear to force the SA algorithm into local minima, as opposed to the indication by previous work [9]. This was apparent because with an increase in the use of forced moves, there was a steady increase in the performance, not a decline after 70% forced moves as expected. This may be due to the technique used in the implementation which contained a large amount of randomness while still constraining components towards the center of their nets. Swaps were especially enhanced with their modification, more than tripling the acceptance rate for many designs.

While the speed of the software implementation has been improved, it does not produce

results that are on par with the other tested tools, even using the force directed technique. The min-cut algorithm currently takes a prohibitive amount of runtime compared to the decrease of wire length that it offered.

The estimation of division done in hardware should not be a hindrance to the algorithm. The results were inside the tolerance that might be expected from a change in the random seed, that is, around plus or minus five percent. The speedup generated by the VLSI model indicates that the cost calculation is no longer the slowest part of the SA process. More portions of the SA process may be put into hardware as the slowest component changes. Currently results show that the slowest part of the algorithm is the move rejection time. This part is responsible for undoing a move if it has been rejected.

5.2 Future work

5.2.1 Software implementation

The min-cut algorithm does improve the results of the Simulated Annealing application, however the overhead involved does not currently seem to justify the slim benefit of the use. No significantly faster min-cut algorithm currently exists, so any improvement in this section must be by improving the gain from using the min-cut. Greater adaptability of the min-cut algorithm to determine groups is required. For example, the current min-cut algorithm tends towards groups of equal size; however a design may have two clear groups of drastically different sizes. For example, a state machine is bound to be much smaller than the logic it controls.

The row-based legalization function needs to be expanded to deal with rows which are over-constrained. It is very important that the results of a Simulated Annealing run are legal. It is unfortunate if they must be thrown out for a trivial matter, such as an over-utilized first row and an under-utilized second row. Many designs incorporate standard cells which are more than one row in height. These cells should be supported in order to support more benchmarks and circuits.

As designs get larger, typically more cells are on each row. Iterating across rows (which are linked lists) seems to be becoming tedious as the designs grow. If method for quickly locating a position in a row was developed, it is expected that the efficiency of the row-based design would improve.

The use of hardware for creating perturbations should be constructed to further lower the requirement of a general purpose processor. Since the vast majority of perturbations that are created are rejected, the ability to create a perturbation then reject it as quickly as possible becomes desirable. Once this has been completed, along with acceptance logic, randomization logic, and glue logic, a general purpose processor may be able to act as an arbiter and file system interface only. Further speedup may later be realized by multiple perturbations being created and scored simultaneously.

Bibliography

[1] C. Alpert. The ISPD98 circuit benchmark suite. *International Symposium on Physical Design*, pages 80–85, April 1998.

[2] William Batts Jr. Modeling of a hardware vlsi placement system: Accelerating the simulated annealing algorithm. *Master's Thesis, Computer Engineering Department, RIT*, 2005.

[3] Carlos A. Coello and David A. Van Veldhuizen nad Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2002.

[4] Sumit Dasgupta and Iain Farquharson. Cadence lef/def exchange format. Web resource: http://openeda.si2.org/projects/lefdef/.

[5] Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing*. Hyperion Books, 1987.

[6] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[7] Olle Haggstrom. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.

[8] R. Hall. Moore's law 40th anniversary.
Web resource: www.intel.com/pressroom/kits/events/moores_law_40th.

[9] Renato Fernandes Hentschke and Richardo Augusto da Luz Reis. Improving simulated annealing placement by applying random and greedy mixed perturbations. *Southern Building Code Conference International*, 2003.

[10] International Symposium on Circuits And Systems. *The ISCAS '85 benchmark circuits and netlist format*, 1985.

[11] S. Kirkpatrik, C. D. Gellatt Jr., and M. P.Vecchi. Optimization by simulated annealing. *Science*, 220(4598), May 1983.

[12] J.M. Kleinhans, G. Sigl, F.M. Johannes, and K.J. Antreich. Vlsi placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(3):356–365, Mar 1991.

[13] Ken Martin. *Digital Integrated Circuit Design*. Oxford University Press, USA, 2006.

[14] N. Quinn and M. Breuer. A forced directed component placement procedure for printed circuit boards. *Circuits and Systems, IEEE Transactions on*, 26(6):377–388, Jun 1979.

[15] Carl Sechen and Alberto Sangovanni-Vincentelli. The timberwolf placement and routing package. *IEEE Journal of Solid-State Circuits*, sc-20(2), Apr 1985.

[16] Mechthild Stoer and Frank Wagner. Simple min-cut algorithm. *Association for Computing Machinery*, 44(4):585–591, July 1997.

[17] Lixin Su, Wray Buntine, A. Richard Newton, and Bradley S. Peters. Learning as applied to stochastic optimization for standard-cell placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(4), Apr 2001.

[18] Symposium on Physical Design. *Capo: Robust and Scalable Open-Source Min-cut Floorplacer*, 2005.

[19] Natarajan Viswanathan and Chris Chong-Nuen Chu. FastPlace: efficient analytical placement using cell shifting, iterative local refinement,and a hybrid net model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(5):722–733, May 2005.

[20] Maogang Wang, Xiaojian Yang, and Majid Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. *International Conference on Computer-Aided Design*, 00:260, 2000.

[21] Eric W. Weisstein. "Mincut." From Mathworld–A Wolfram Web Resource. Web resource: mathworld.wolfram.com/Mincut.html.

[22] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr 1997.